APPENDIX

ACTIVITY 11.01: CREATING A CONTACT STUB AND TEST

Solution:

1. Create a class called **`ContactStub.php`** inside the **`app`** folder:

```
<?php
namespace App;
class ContactStub
{
    public function getContact()
    {
        // Do something.
    }
}
```

   This has only one method, **`getContact`**, which returns nothing. That's fine, as its value will be a canned response in the implementation.

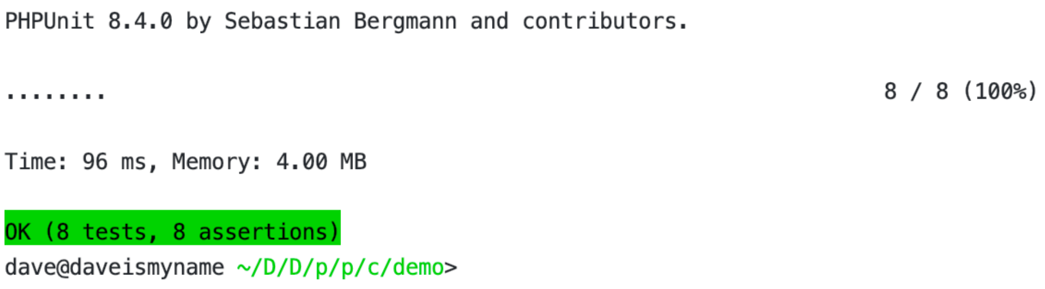
2. Create a test called **`ContactStubTest.php`** inside the **`tests`** folder. By using the **`$this->createMock()`** method, provide the path to the stub to be used. **`then ->method()`** is used to specify the method from the given stub and **`finally ->willReturn()`** is used to specify the return value of the stub. This can then be used to assert that a contact will be returned from the stub and will match a fixed input, in this case – **`Joe Bloggs`**:

```
<?php
use PHPUnit\Framework\TestCase;
class ContactStubTest extends TestCase
{
    /** @test */
    public function willStubContact()
    {
        $stub = $this->createMock(App\ContactStub::class);
        $stub->method('getContact')
            ->willReturn('Joe Bloggs');
        $expected = 'Joe Bloggs';
        $actual = $stub->getContact();
        $this->assertEquals($expected, $actual);
    }
}
```

3. Run the test as follows:

```
vendor/bin/phpunit
```

The test passes as the fixed return value matched the expected value:



```
PHPUnit 8.4.0 by Sebastian Bergmann and contributors.

........                                                            8 / 8 (100%)

Time: 96 ms, Memory: 4.00 MB

OK (8 tests, 8 assertions)
dave@daveismyname ~/D/D/p/p/c/demo>
```

Figure 11.23: Fixed return value matches the expected value

This allows a real object to be replaced with a stub so that canned responses can be tested against. This will be more reliable than using an external resource.